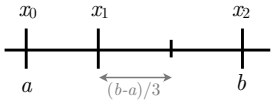
Homework 6 Selected Solutions
APPM/MATH 4650 Fall '20 Numerical Analysis

Due date: Saturday, October 17, before midnight, via Gradescope. **Instructor**: Prof. Becker
Theme: Numerical integration

solutions version 10/9/2020

Problem 1: Quadrature on non-equispaced nodes Consider (non-composite) quadrature to estimate $\int_a^b f(x)\,dx$ using nodes $x_0 = a$, $x_1 = a + \frac{1}{3}(b-a)$, $x_2 = b$. See the figure below:



Determine a quadrature scheme using these nodes that can integrate quadratic polynomials exactly (i.e., has degree of accuracy 2). *Hint:* Without loss of generality, you can write $h = (b-a)/3$ and then let $x_0 = 0$. *Hint:* Interpolate! *Hint:* Re-use some of the work from last week's HW. *Hint:* Check your work numerically by seeing if your scheme really does integrate a quadratic exactly.

Solution:

Without loss of generality, let $x_0 = 0$, so our nodes are $\{x_0 = 0, x_1 = h, x_2 = 3h\}$. The interpolating polynomial is

$$p(x) = f(0)\ell_0(x) + f(h)\ell_1(x) + f(3h)\ell_2(x)$$

where

$$\ell_0(x) = \frac{(x-h)(x-3h)}{(-h)(-3h)} = \frac{1}{3h^2}\left(x^2 - 4hx + 3h^2\right)$$
$$\ell_1(x) = \frac{(x)(x-3h)}{(h)(h-3h)} = -\frac{1}{2h^2}\left(x^2 - 3hx\right)$$
$$\ell_2(x) = \frac{(x)(x-h)}{(3h)(3h-h)} = \frac{1}{6h^2}\left(x^2 - hx\right)$$

since the formula for the i^{th} Lagrange polynomial of degree n is $\ell_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$. *Note that this is the same calculation as in last week's homework, so you can reuse all your work!*

We calculate

$$\int_0^{3h} \ell_0(x) = \frac{1}{3h^2}\left(\frac{1}{3}x^3 - 2hx^2 + 3h^2x\right)\Big|_{x=0}^{3h} = \frac{1}{3h^2}\left(\frac{27}{3}h^3 - 18h^3 + 9h^3\right) = 0$$
$$\int_0^{3h} \ell_1(x) = -\frac{1}{2h^2}\left(\frac{1}{3}x^3 - \frac{3}{2}hx^2\right)\Big|_{x=0}^{3h} = -\frac{1}{2h^2}\left(\frac{27}{3}h^3 - \frac{27}{2}h^3\right) = \frac{9}{4}h$$
$$\int_0^{3h} \ell_2(x) = \frac{1}{6h^2}\left(\frac{1}{3}x^3 - \frac{1}{2}hx^2\right)\Big|_{x=0}^{3h} = \frac{1}{6h^2}\left(\frac{27}{3}h^3 - \frac{9}{2}h^3\right) = \frac{3}{4}h.$$

Then we find our estimate for $\int_0^{3h} f(x)\,dx$ by computing $\int_0^{3h} p(x)\,dx$, which is

$$\int_0^{3h} p(x)\,dx = f(0)\int_0^{3h} \ell_0(x)\,dx + f(h)\int_0^{3h} \ell_1(x)\,dx + f(3h)\int_0^{3h} \ell_2(x)\,dx$$
$$= \boxed{h\left(0f(0) + \frac{9}{4}f(h) + \frac{3}{4}f(3h)\right)}.$$

Because we interpolate with a degree 2 polynomial, if f is a polynomial of degree 2, then since p is the *unique* degree 2 interpolating polynomial, it follows $f = p$, and hence this formula is exact. It's not yet clear if the degree of accuracy is 3 (for Simpson's rule, with equispaced nodes, it just so happens that the degree of accuracy is 3, but recall that equispaced nodes with n even, we don't get so lucky). To check that, we'd need to do a Taylor expansion or use the error formula for interpolating polynomials (Thm. 3.3 in Burden and Faires).

Problem 2: Make a composite version of the scheme you developed in Problem 1, and use it to integrate $f(x) = e^{3x}$ on $[-1, 2]$. Plot the error as a function of the number of nodes n (choose a large value of n, like 10^2 or 10^4; you may want to make sure n is a multiple of a small number); as always, we suggest `logspace` here. Also plot the error using composite midpoint, trapezoidal and Simpson's rule. Using the plot, estimate the order of the new composite rule you created.

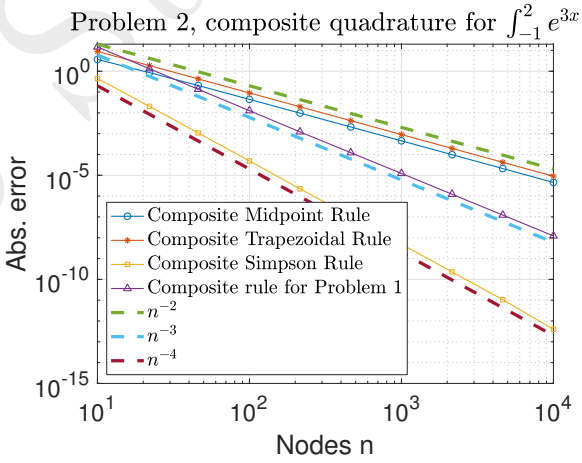
Solution:

First, we make sure n is divisible by 3, then we make a grid of $n+1$ nodes from $[a, b]$ (inclusive) labeled $\{x_0, x_1, \ldots, x_n\}$. Then repeating the quadrature formula from Problem 1 n times gives

$$h\left(\frac{9}{4}\sum_{i=1,4,7,\ldots}^{n-3} f(x_i) + \frac{3}{4}\sum_{i=3,6,9,\ldots}^{n} f(x_i)\right), \quad h = \frac{b-a}{n}$$

Note that we only use 0-based indices $\{1, 4, 7, \ldots\}$ and $\{3, 6, 9, \ldots\}$ so in fact we could have made half as many nodes in the first place since we never evaluate f at all of the nodes we made. Either convention is OK.

The plot is



From the plot, we estimate that the composite rule we made is $\boxed{O(n^{-3})}$.

The basic code is below; see the end of the PDF for full code including `CompositeQuad.m`

```
f = @(x) exp(3*x);
F = @(x) 1/3*exp(3*x);
a = -1;
b = 2;
I = F(b) - F(a);
```

```
mid  = @(n) CompositeQuad( f, a, b, n, 'midpoint');
trap = @(n) CompositeQuad( f, a, b, n, 'trapezoidal');
simp = @(n) CompositeQuad( f, a, b, n, 'simpson');
probl1 = @(n) CompositeQuad( f, a, b, n, 'problem1');

nList = round(logspace(1,4,10));
[errMidpoint,errTrapezoidal,errSimpson,errProblem1] = deal([]);
for n = nList
  errMidpoint(end+1)      = abs( mid(n)  - I );
  errTrapezoidal(end+1)   = abs( trap(n) - I );
  errSimpson(end+1)       = abs( simp(n) - I );
  errProblem1(end+1)      = abs( probl1(n) - I );
end
```

Problem 3: Romberg integration Make a Romberg integration scheme applied to the composite *midpoint* rule, and evaluate this to approximate $\int_0^{50} \cos(x)\,dx$ using up to $n = 4096$. Report a table of your error with each row a larger value of n and each column the next extrapolation value for that n (e.g., similar to Tables 4.10 and 4.11 in Burden and Faires, though note these are for composite *trapezoidal* rule). Both composite midpoint and composite trapezoidal rules are $O(h^2)$ (assuming f has bounded derivatives), but is there an advantage to using composite trapezoidal rule in Romberg integration? *Note*: Your code does *not* need to efficiently re-use function evaluations on nodes.

Solution:

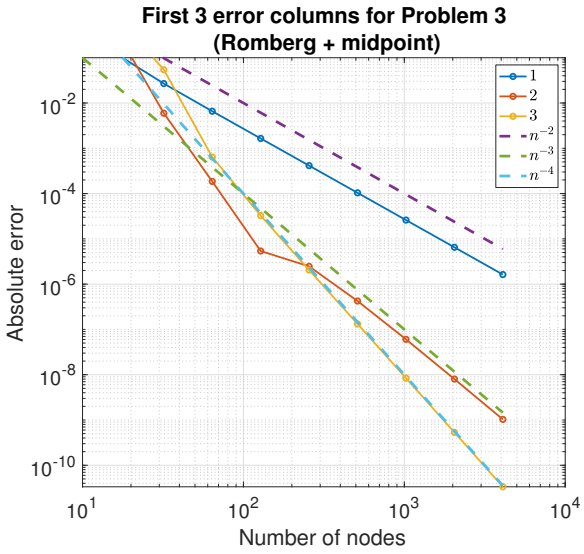
```
   4 +1.6e+00
   8 -1.8e+00 -2.9e+00
  16 -1.3e-01 +4.3e-01 +9.1e-01
  32 -2.7e-02 +6.0e-03 -5.4e-02 -1.2e-01
  64 -6.6e-03 +1.9e-04 -6.4e-04 +2.9e-03 +6.9e-03
 128 -1.6e-03 -5.4e-06 -3.3e-05 +7.8e-06 -8.7e-05 -2.0e-04
 256 -4.1e-04 -2.5e-06 -2.1e-06 -1.2e-08 -2.6e-07 +1.1e-06 +2.7e-06
 512 -1.0e-04 -4.2e-07 -1.3e-07 -3.9e-09 -3.7e-09 +4.5e-10 -8.3e-09 -1.9e-08
1024 -2.6e-05 -6.0e-08 -8.4e-09 -1.8e-10 -6.1e-11 -3.7e-12 -7.3e-12 +2.5e-11 +6.2e-11
2048 -6.5e-06 -8.0e-09 -5.3e-10 -6.6e-12 -1.0e-12 -5.4e-14 -2.5e-14 +2.9e-15 -4.7e-14 -1.1e-13
4096 -1.6e-06 -1.0e-09 -3.3e-11 -2.3e-13 -2.0e-14 -4.8e-15 -4.4e-15 -4.3e-15 -4.3e-15 -4.3e-15 -4.2e-15
```

Part of the Matlab code to do this is below (the rest, including `Romberg_generic`, is at the end of the PDF)

```
f = @(x) cos(x);
F = @(x) sin(x);
a = 0;
b = 50;
I = F(b) - F(a);
mid  = @(n) CompositeQuad( f, a, b, n, 'midpoint');
[nList,table] = Romberg_generic( 2^12, mid, (2:20) );
```

The main thing we're looking for is that you understand the error looks like $O(h^2) + O(h^3) + O(h^4)\ldots$ (in comparison, for composite trapezoidal, it looks like $O(h^2)+O(h^4)+O(h^6)\ldots$, and for composite Simpson's it looks like $O(h^4) + O(h^6) + O(h^8)\ldots$). You can figure this out via trial-and-error, and plotting some columns of the table may help (use logarithmic scaling for both axes).

We can see by plotting the first 3 columns of the table (excluding the first column which is



just n)

Then to answer our question, even though both composite midpoint and composite trapezoidal are $O(h^2)$, because composite midpoint is $O(h^2)+O(h^3)+O(h^4)\ldots$ but composite trapezoidal is $O(h^2)+O(h^4)+O(h^6)\ldots$, this means that Romberg integration for composite midpoint doesn't work quite as well as it does for composite trapezoidal rule. That is, for Romberg with midpoint, the columns decay at $O(h^2)+O(h^3)+O(h^4)\ldots$, whereas for composite trapezoidal they decay at $O(h^2)+O(h^3)+O(h^4)\ldots$.

Problem 4: Fourier things Consider a function f which is periodic on $[-\pi,\pi]$. We define the k^{th} **Fourier coefficient** (for $k\in\mathbb{Z}$, where $\mathbb{Z}$ are the integers) as

$$\hat{f}_k = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)e^{-ikx}\,dx \tag{1}$$

which are then used to represent f as a **Fourier series** $f(x)=\sum_{k\in\mathbb{Z}}\hat{f}_k e^{ikx}$. If you haven't seen Fourier coefficients, they are similar to the **Fourier transform**[1] but for periodic functions; if you haven't seen the Fourier transform, don't worry as you don't need a familiarity with it for this problem, but you ought to see the Fourier transform at some point before you graduate (if you're in a STEM field). Yet another related concept is the **discrete Fourier transform (DFT)** (which is what the FFT computes):

$$F_k = \frac{1}{n}\sum_{j=-n/2+1}^{n/2} f_j e^{-i2\pi jk/n}, \quad k=-n/2+1,-n/2+2,\ldots,0,\ldots n/2-1,n/2 \tag{2}$$

where $f_j=f(j\cdot h)$ with $h=2\pi/n$ (note we use $i=\sqrt{-1}$, and j is just an integer index)[2]. The DFT is useful in its own right, but it can also be viewed as an approximation to the Fourier coefficients for $|k|\le n/2$. For this problem, we'll assume n is even to simplify indexing, though similar formulas hold if n is odd as well.

a) Assuming f is 2π periodic so $f(-\pi)=f(\pi)$, and for $|k|\le n/2$, show that the DFT in Eq. (2) is the composite trapezoidal rule (with $n+1$ nodes) applied to the integral for the Fourier coefficients in Eq. (1).

[1] which is "defined" as $\hat{f}(\omega)=\int_{\mathbb{R}} f(x)e^{-ikx}\,dx$ for any $\omega\in\mathbb{R}$ if $\int_{\mathbb{R}}|f|\,dx$ exists (this is the "definition" you learn in a physics or engineering class), though you can also define the Fourier transform on a larger class of functions by thinking of it as a bounded linear transformation and using density arguments, or by using in a weak sense (using test functions) so that you can even make sense of the Fourier transform of a Dirac delta function; such results are discussed in APPM 5450 "Applied Analysis." Engineers and physicists often implicitly use this extended definition.

[2] For all these transforms, there are many different conventions for the normalization constants, and in the end it's not too important as long as you are consistent, which usually means adjusting the appropriate *inverse* transform.

Solution:

The composite trapezoidal rule applied to Eq. (1) using the $n+1$ equispaced nodes $\{x_j\}_{j=-n/2}^{n/2}$ where $x_j = jh$, $x_{-n/2} = -\pi$ and $x_{n/2} = \pi$ and $h = \frac{2\pi}{n}$, is

$$\begin{aligned}
&\frac{h}{2}\left(\frac{1}{2\pi}f(x_{-n/2})e^{-ikx_{-n/2}} + 2\sum_{j=-n/2+2}^{n/2-1}\frac{1}{2\pi}f(x_j)e^{-ikx_j} + \frac{1}{2\pi}f(x_{n/2})e^{-ikx_{n/2}}\right)\\
&= \frac{h}{2}\left(2\sum_{j=-n/2+2}^{n/2-1}\frac{1}{2\pi}f(x_j)e^{-ikx_j} + 2\frac{1}{2\pi}f(x_{n/2})e^{-ikx_{n/2}}\right) \quad \text{since } f(-\pi) = f(\pi)\\
&= \frac{2\pi}{2n}\left(2\sum_{j=-n/2+2}^{n/2-1}\frac{1}{2\pi}f(x_j)e^{-ikx_j} + \frac{1}{2\pi}f(x_{n/2})e^{-ikx_{n/2}}\right)\\
&= \frac{1}{n}\sum_{j=-n/2+2}^{n/2} f(x_j)e^{-ikx_j}\\
&= \frac{1}{n}\sum_{j=-n/2+2}^{n/2} f_j e^{-ik2\pi j/n}
\end{aligned}$$

which is F_k from Eq. (2), as desired.

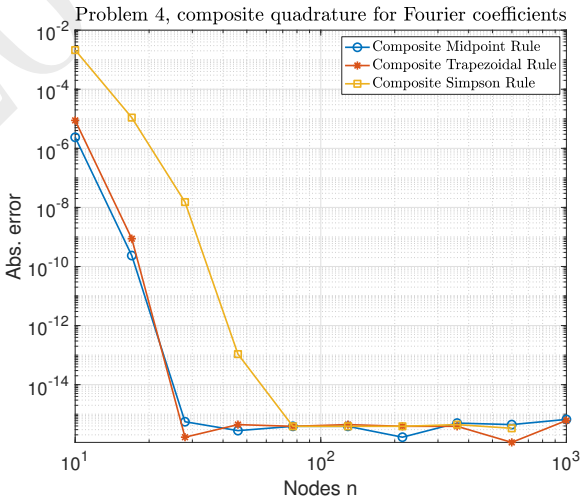
b) Let $f(x) = \frac{1}{1+\cos(x)/2}$, for which the Fourier coefficients can be evaluated in closed form using complex variables and the theory of residues (or via Mathematica, Wolfram Alpha, or sympy). For example,

$$\hat{f}_1 = \frac{1}{2\pi}\int_{-\pi}^{\pi}\frac{e^{-ikx}}{1+\cos(x)/2}\,dx = 2 - \frac{4}{\sqrt{3}}$$

Write code to estimate the value of $\hat{f}_1$ using composite midpoint, trapezoidal and Simpson's rule, and plot the error for each method as a function of n (for using $n+1$ nodes), for reasonable values of n.

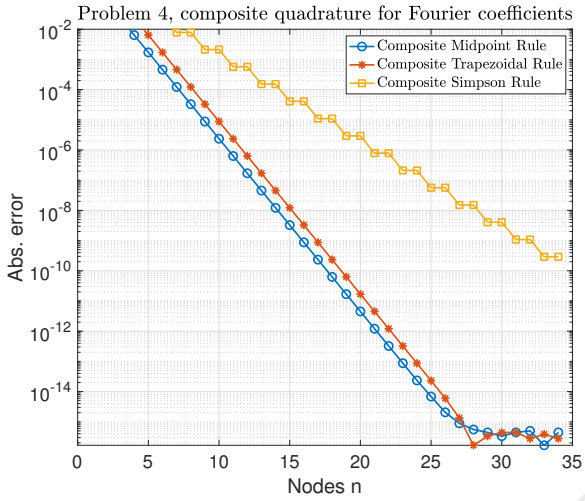
Solution:

The code is straightforward, as we can reuse the code we wrote for the previous problems. A plot of the error is below:



In fact, because the error converges so fast (better than $n^{-\alpha}$ for any constant α), we get better insight by keeping the x-axis linear scale (for HW submissions, either kind of

scaling for the x-axis is OK, but the y-axis *must* be logarithmic, otherwise we get almost no useful information from the plot).



c) Recalling that the composite trapezoidal rule is the same as the DFT formula, based on your findings in part (b), do you think there's an advantage to using a higher-order method, like composite Simpson's rule, to approximate the Fourier coefficient? Give some theoretical evidence behind your answer.[3]

Solution:

In this case, the midpoint and trapezoidal rules (whose formulas are nearly identical in this case, except for the spacing of the nodes) both give better-than-expected error, and in fact drop to machine precision by about $n = 30$. Simpson's rule does not do as well, though it does converge very quickly. Thus in this case, there is no benefit (and in fact it's slightly worse) to use the higher-order Simpson's rule to approximate the Fourier coefficient.

We can speculate about why this is by looking at the Euler-Maclaurin formula, which puts the error in terms of $f^{(k)}(b) - f^{(k)}(a)$ for various orders of derivatives k, and where we use $a = -\pi$ and $b = \pi$ for this problem. Since f is period, these derivative terms cancel. To make a more precise statement (e.g., Thm. 9.3 in Briggs and Henson) we would need to control an error term (another derivative term); see theorem below. Students got full credit if they mentioned something about Euler-Maclaurin (they did not need to elaborate or go into details).

A precise statement of Euler-Maclaurin, take from Briggs and Henson, is below (note that what they call N is what we called n, and they instead use n as an index much like we used j):

[3] For further reading on the connections between the DFT and quadrature, see chapter 9 in *The DFT: An Owner's Manual for the Discrete Fourier Transform* by William Briggs and Van Emden Henson, SIAM (1995); the function used in this exercise is based on the similar function on page 366 of that book. For a precise statement on when there is no error at all in evaluating the Fourier coefficient via the DFT, see Theorem 9.3 in that book.

THEOREM 9.2. EULER–MACLAURIN SUMMATION FORMULA. *Let $g \in C^{2p+2}[-\pi,\pi]$ for $p \geq 1$, and let $T_N\{g\}$ be the trapezoid rule approximation to $\int_{-\pi}^{\pi} g(x)dx$ with N uniform subintervals. Then the error in this approximation is*

$$\begin{aligned} E_N &= T_N\{g\} - \int_{-\pi}^{\pi} g(x)dx \\ &= \sum_{m=1}^{p} \left\{ \left(\frac{2\pi}{N}\right)^{2m} \frac{B_{2m}}{(2m)!} \left[g^{(2m-1)}(\pi) - g^{(2m-1)}(-\pi)\right] \right. \\ &\qquad \left. + 2\pi \left(\frac{2\pi}{N}\right)^{2p+2} \frac{B_{2p+2}}{(2p+2)!} g^{(2p+2)}(\xi) \right\}, \end{aligned}$$

where $-\pi < \xi < \pi$, and B_n is the nth Bernoulli number (to be discussed shortly).

Theorem 9.3 from Briggs and Henson is as follows:

Theorem 9.3 "Zero error in the trapezoid rule." *Assume $f \in C^{\infty}[-\pi,\pi]$ and that f and and all of its derivatives are 2π-periodic. Let $|f^{(p)}(x)| \leq \alpha^p$ on $[-\pi,\pi]$ for some $\alpha > 0$ and for all $p \geq 0$. Then the trapezoid rule (DFT) is exact when used to approximate the Fourier coefficient*

$$c_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) e^{-ikx}\, dx,$$

provided that $\alpha < |k| < N$.

The other main tool to bound the error in the DFT is the *Poisson summation formula*, which we will not discuss.

Optional problem (Not graded) **Beating Mathematica**. Think of a complicated function F, and then symbolically find the derivative $f = F'$ (using your favorite symbolic math software, like Wolfram Alpha, Mathematica, python with sympy, Matlab with the symbolic package, sage, Maple, etc). Try to make F (and hence f) so complicated that the software *cannot* find the antiderivative of f (or at least not in a reasonable amount of time). Then numerically evaluate the integral of f over an appropriate range $[a,b]$ using your favorite quadrature rule, and check the error (since you know the true value $\int_a^b f(x)\,dx = F(b) - F(a)$).[4]

Problem 2 complete code

Here is the code for Problem 2:

```
function I = CompositeQuad( f, a, b, n, method )
% I = CompositeQuad( f, a, b, n, method )
%   where method is one of 'trapezoid','midpoint', or 'simpson'
%   (case insensitive) returnsn the composite quadrature rule
%   approximation of \int_a^b f(x) dx using n+1 nodes.
switch lower(method)
  case {1,'mid','midpoint','midpt'} % any of these are valid
    h   = (b-a)/(n+1);
    nodes   = (a+h/2):h:b;
    I       = h*sum(f(nodes));
  case {0,'trap','t','trapezoid','trapezoidal'}
    h       = (b-a)/n;
    nodes   = linspace(a,b,n+1);
```

[4] In the case of Mathematica, we haven't really *beaten Mathematica*, just its `Integrate` function, since Mathematica includes quadrature-based numerical integration in `NIntegrate`.

```
    y       = f(nodes);
    I       = h*(sum(y) - y(1)/2  - y(end)/2 );
  case {2,'simp','simpson','simpsons'}
    n       = max(2,round(n/2)*2); % make sure its even
    h       = (b-a)/n;
    nodes   = linspace(a,b,n+1);
    y       = f(nodes);
    I       = h/3*( y(1) + 2*sum(y(3:2:end-1)) + ...
    4*sum(y(2:2:end-1)) + y(end) );
  case 'problem1'
    n       = max(3,round(n/3)*3); % make it divisible by 3
    h       = (b-a)/n;
    nodes   = linspace(a,b,n+1);
    y       = f(nodes);
    % there's no double-counting! Use h, not h/3
    I       = h*( 9/4*sum(y(2:3:end-2)) + 3/4*sum(y(4:3:end)) );
  otherwise
    error('CompsiteQuad:badMethod','Did not recognize method type');
end
if ~isreal(I)
  if abs(imag(I))/abs(real(I)) < 1e-8
    I = real(I);
  end
end
```

and the plotting code was

```
loglog(nList,errMidpoint,'o-','linewidth',1,'DisplayName','Composite Midpoint Rule');
hold all
loglog(nList,errTrapezoidal,'*-','linewidth',1,'DisplayName','Composite Trapezoidal Rule');
loglog(nList,errSimpson,'s-','linewidth',1,'DisplayName','Composite Simpson Rule');
loglog(nList,errProblem1,'^-','linewidth',1,'DisplayName','Composite rule for Problem 1');
loglog(nList, 2e3./nList.^2, '--', 'linewidth',3,'DisplayName','$n^{-2}$');
loglog(nList, 6e3./nList.^3, '--', 'linewidth',3,'DisplayName','$n^{-3}$');
loglog(nList, 2e3./nList.^4, '--', 'linewidth',3,'DisplayName','$n^{-4}$');
grid on
set(gca,'fontsize',22);
legend('location','southwest','interpreter','latex','fontsize',18)
xlabel('Nodes n'); ylabel('Abs. error');
title('Problem 2, composite quadrature for $\int_{-1}^2e^{3x}$','interpreter','latex')
export_fig HW6_problem2 -pdf -transparent
```

Problem 3 complete code

Here is a more complete version of Problem 3:

```
function [nList,table] = Romberg_generic( N, quadRule, ordersOfRule )
% [nList,table] = Romberg_generic( N, quadRule, ordersOfRule )
%   makes the table output for Romberg integration up to N nodes
%   using the quadrature rule quadRule which should be a function
%   of the number of nodes.  ordersOfRule should be a list
%   specifying the orders of h of the quadrature rule,
%   e.g., for composite trapezoidlal this is [2,4,6,...]
n = max(4,2^(nextpow2(N)-length(ordersOfRule)) );
table = [];
```

```
nList = [];
while n <= N
I = quadRule(n);
if isempty(table)
  table = I;
else
  % Make a new row to append
  oldRow  = table(end,:);
  J       = length( oldRow );
  newRow  = zeros(1,J+1);
  newRow(1) = I;
  for j = 1:J
    qj = 2^ordersOfRule(j);
    % newRow(j+1) = newRow(j) + (newRow(j)-oldRow(j))/( qj - 1 );
    % re-arrange the above (in some cases, this is more stable)
    newRow(j+1) = ( qj*newRow(j) -oldRow(j))/( qj - 1 );
  end
  table = [table, zeros(J,1); newRow ]; % append row
end
nList(end+1) = n;
n = 2*n;
end
```

and the plotting/display code for Problem 3 was

```
tableDisplay = (table + triu(I*ones(size(table)),1) )-I;
[nList', tableDisplay]
% Print it nicely for latex
for i = 1:length(nList)
    fprintf('%4d ', nList(i));
    fprintf('%+5.1e ', tableDisplay(i,1:i) ); % abs value OK too
    fprintf('\n');
end
% Optional plot
figure(1); clf;
for i = 1:3
  loglog( nList(i:end), abs(table(i:end,i)-I), 'o-','DisplayName',num2str(i),'linewidth',2);
  hold all
end
loglog( nList, 1e2./nList.^2, '--','linewidth',3,'DisplayName','$n^{-2}$');
loglog( nList, 1e2./nList.^3, '--','linewidth',3,'DisplayName','$n^{-3}$');
loglog( nList, 1e4./nList.^4, '--','linewidth',3,'DisplayName','$n^{-4}$');
grid on
legend('interpreter','latex','fontsize',18)
set(gca,'fontsize',22);
xlabel('Number of nodes');
ylabel('Absolute error');
xlim([10,1e4]);
ylim([-Inf,1e-1]);
title(sprintf('First 3 error columns for Problem 3\n (Romberg + midpoint)'));
export_fig 'HW6_problem3' -pdf -transparent
```